INSTRUCTION FETCH APPARATUS FOR WIDE ISSUE PROCESSORS AND METHOD OF OPERATION

CROSS-REFERENCE TO RELATED APPLICATIONS

The present invention is related to those disclosed in the following United States Patent Applications:

1) Serial No. [Docket No. 00-BN-051], filed concurrently herewith, entitled "SYSTEM AND METHOD FOR EXECUTING VARIABLE LATENCY LOAD OPERATIONS IN A DATA PROCESSOR";

2) Serial No. [Docket No. 00-BN-052], filed concurrently herewith, entitled "PROCESSOR PIPELINE STALL APPARATUS AND METHOD OF OPERATION";

3) Serial No. [Docket No. 00-BN-053], filed concurrently herewith, entitled "CIRCUIT AND METHOD FOR HARDWARE-ASSISTED SOFTWARE FLUSHING OF DATA AND INSTRUCTION CACHES";

4) Serial No. [Docket No. 00-BN-054], filed concurrently herewith, entitled "CIRCUIT AND METHOD FOR SUPPORTING MISALIGNED ACCESSES IN THE PRESENCE OF SPECULATIVE LOAD INSTRUCTIONS";

5) Serial No. [Docket No. 00-BN-055], filed

concurrently herewith, entitled "BYPASS CIRCUITRY FOR USE IN A PIPELINED PROCESSOR";

6) Serial No. [Docket No. 00-BN-056], filed concurrently herewith, entitled "SYSTEM AND METHOD FOR EXECUTING CONDITIONAL BRANCH INSTRUCTIONS IN A DATA PROCESSOR";

7) Serial No. [Docket No. 00-BN-057], filed concurrently herewith, entitled "SYSTEM AND METHOD FOR ENCODING CONSTANT OPERANDS IN A WIDE ISSUE PROCESSOR";

8) Serial No. [Docket No. 00-BN-058], filed concurrently herewith, entitled "SYSTEM AND METHOD FOR SUPPORTING PRECISE EXCEPTIONS IN A DATA PROCESSOR HAVING A CLUSTERED ARCHITECTURE";

9) Serial No. [Docket No. 00-BN-059], filed concurrently herewith, entitled "CIRCUIT AND METHOD FOR INSTRUCTION COMPRESSION AND DISPERSAL IN WIDE-ISSUE PROCESSORS";

10) Serial No. [Docket No. 00-BN-066], filed concurrently herewith, entitled "SYSTEM AND METHOD FOR REDUCING POWER CONSUMPTION IN A DATA PROCESSOR HAVING A CLUSTERED ARCHITECTURE".

The above applications are commonly assigned to the assignee

of the present invention. The disclosures of these related patent applications are hereby incorporated by reference for all purposes as if fully set forth herein.

TECHNICAL FIELD OF THE INVENTION

The present invention is generally directed to data processors and, more specifically, to an efficient instruction fetch engine for use in a wide issue data processor.

BACKGROUND OF THE INVENTION

The demand for high performance computers requires that state-of-the-art microprocessors execute instructions in the minimum amount of time. A number of different approaches have been taken to decrease instruction execution time, thereby increasing processor throughput. One way to increase processor throughput is to use a pipeline architecture in which the processor is divided into separate processing stages that form the pipeline. Instructions are broken down into elemental steps that are executed in different stages in an assembly line fashion.

A pipelined processor is capable of executing several different machine instructions concurrently. This is accomplished by breaking down the processing steps for each instruction into several discrete processing phases, each of which is executed by a separate pipeline stage. Hence, each instruction must pass sequentially through each pipeline stage in order to complete its execution. In general, a given instruction is processed by only one pipeline stage at a time, with one clock cycle being required for each stage. Since instructions use the pipeline stages in the same order and typically only stay in each stage for a single clock cycle, an N stage pipeline is capable of simultaneously processing

N instructions. When filled with instructions, a processor with N pipeline stages completes one instruction each clock cycle.

The execution rate of an N-stage pipeline processor is theoretically N times faster than an equivalent non-pipelined processor. A non-pipelined processor is a processor that completes execution of one instruction before proceeding to the next instruction. Typically, pipeline overheads and other factors decrease somewhat the execution rate advantage that a pipelined processor has over a non-pipelined processor.

An exemplary seven stage processor pipeline may consist of an address generation stage, an instruction fetch stage, a decode stage, a read stage, a pair of execution (E1 and E2) stages, and a write (or write-back) stage. In addition, the processor may have an instruction cache that stores program instructions for execution, a data cache that temporarily stores data operands that otherwise are stored in processor memory, and a register file that also temporarily stores data operands.

The address generation stage generates the address of the next instruction to be fetched from the instruction cache. The instruction fetch stage fetches an instruction for execution from the instruction cache and stores the fetched instruction in an instruction buffer. The decode stage takes the instruction from

the instruction buffer and decodes the instruction into a set of signals that can be directly used for executing subsequent pipeline stages. The read stage fetches required operands from the data cache or registers in the register file. The E1 and E2 stages perform the actual program operation (e.g., add, multiply, divide, and the like) on the operands fetched by the read stage and generates the result. The write stage then writes the result generated by the E1 and E2 stages back into the data cache or the register file.

Assuming that each pipeline stage completes its operation in one clock cycle, the exemplary seven stage processor pipeline takes seven clock cycles to process one instruction. As previously described, once the pipeline is full, an instruction can theoretically be completed every clock cycle.

The throughput of a processor also is affected by the size of the instruction set executed by the processor and the resulting complexity of the instruction decoder. Large instruction sets require large, complex decoders in order to maintain a high processor throughput. However, large complex decoders tend to increase power dissipation, die size and the cost of the processor. The throughput of a processor also may be affected by other factors, such as exception handling, data and instruction cache

sizes, multiple parallel instruction pipelines, and the like. All of these factors increase or at least maintain processor throughput by means of complex and/or redundant circuitry that simultaneously increases power dissipation, die size and cost.

In many processor applications, the increased cost, increased power dissipation, and increased die size are tolerable, such as in personal computers and network servers that use x86-based processors. These types of processors include, for example, Intel Pentium™ processors and AMD Athlon™ processors.

However, in many applications it is essential to minimize the size, cost, and power requirements of a data processor. This has led to the development of processors that are optimized to meet particular size, cost and/or power limits. For example, the recently developed Transmeta Crusoe™ processor greatly reduces the amount of power consumed by the processor when executing most x86 based programs. This is particularly useful in laptop computer applications. Other types of data processors may be optimized for use in consumer appliances (e.g., televisions, video players, radios, digital music players, and the like) and office equipment (e.g., printers, copiers, fax machines, telephone systems, and other peripheral devices). The general design objectives for data processors used in consumer appliances and office equipment are the

minimization of cost and complexity of the data processor.

Many pipelined processors are implemented as very large instruction word (VLIW) devices that allow the parallel execution of multiple instructions in two or more instruction pipelines. A common problem in VLIW processors is the complexity of the fetch and instruction alignment circuitry. The problem arises because variable numbers of instructions are executed each cycle, making it difficult to decide where to fetch from next. Some prior art solutions require extremely simple algorithms that suffer more stall cycles than necessary. Other prior art solutions use a single point of size encoding (e.g., IA64), which results in a more complex instruction decode circuit and a less flexible issue strategy.

Therefore, there is a need in the art for improved pipeline architectures that allow efficient implementation of very large instruction words (VLIW) in a data processor. In particular, there is a need in the art for an instruction fetch engine that can fetch variable-length very large instruction words from an instruction cache and issue the instructions into an execution pipeline with minimum delay. More particularly, there is a need in the art for an instruction fetch engine that can determine when all portions of a variable-length VLIW have been fetched from an instruction cache

and can issue the complete instruction into an execution pipeline with minimum delay.

SUMMARY OF THE INVENTION

To address the above-discussed deficiencies of the prior art, it is a primary object of the present invention to provide a data processor that implements an instruction issue unit that efficiently transfers instruction bundles from an instruction cache to an instruction execution pipeline with minimum delay. According to an advantageous embodiment of the present invention, the data processor comprises 1) an instruction execution pipeline comprising N processing stages; and 2) an instruction issue unit capable of fetching into the instruction execution pipeline instructions fetched from an instruction cache associated with the data processor, each of the fetched instructions comprising from one to S syllables. The instruction issue unit comprises: a) a first buffer comprising S storage locations capable of receiving and storing the one to S syllables associated with the fetched instructions, each of the S storage locations capable of storing one of the one to S syllables of each fetched instruction; b) a second buffer comprising S storage locations capable of receiving and storing the one to S syllables associated with the fetched instructions, each of the S storage locations capable of storing one of the one to S syllables of each fetched instruction; and c) a

controller capable of determining if a first one of the S storage locations in the first buffer is full, wherein the controller, in response to a determination that the first one of the S storage locations is full, causes a corresponding syllable in an incoming fetched instruction to be stored in a corresponding one of the S storage locations in the second buffer.

According to one embodiment of the present invention, the value of S is four.

According to another embodiment of the present invention, the value of S is eight.

According to still another embodiment of the present invention, the value of S is a multiple of four.

According to a yet another embodiment of the present invention, each of the one to S syllables comprises 32 bits.

According to a further embodiment of the present invention, each of the one to S syllables comprises 16 bits.

According to a still further embodiment of the present invention, each of the one to S syllables comprises 64 bits.

According to a yet further embodiment of the present invention, the controller is capable of determining when all of the syllables in one of the fetched instructions are present in the first buffer, wherein the controller, in response to a

determination that the all of the syllables are present, causes the all of the syllables to be transferred from the first buffer to the instruction execution pipeline.

In one embodiment of the present invention, the controller is capable of determining if a syllable in the first one of the S storage locations in the first buffer has been transferred from the first buffer to the instruction pipeline, wherein the controller, in response to a determination that the first one of the S storage locations has been transferred, causes the corresponding syllable stored in the corresponding one of the S storage locations in the second buffer to be transferred to the first one of the S storage locations in the first buffer.

In another embodiment of the present invention, the data processor further comprises a switching circuit controlled by the controller and operable to transfer syllables from the second buffer to the first buffer.

The foregoing has outlined rather broadly the features and technical advantages of the present invention so that those skilled in the art may better understand the detailed description of the invention that follows. Additional features and advantages of the invention will be described hereinafter that form the subject of the claims of the invention. Those skilled in the art should

appreciate that they may readily use the conception and the specific embodiment disclosed as a basis for modifying or designing other structures for carrying out the same purposes of the present invention. Those skilled in the art should also realize that such equivalent constructions do not depart from the spirit and scope of the invention in its broadest form.

Before undertaking the DETAILED DESCRIPTION OF THE INVENTION below, it may be advantageous to set forth definitions of certain words and phrases used throughout this patent document: the terms "include" and "comprise," as well as derivatives thereof, mean inclusion without limitation; the term "or," is inclusive, meaning and/or; the phrases "associated with" and "associated therewith," as well as derivatives thereof, may mean to include, be included within, interconnect with, contain, be contained within, connect to or with, couple to or with, be communicable with, cooperate with, interleave, juxtapose, be proximate to, be bound to or with, have, have a property of, or the like; and the term "controller" means any device, system or part thereof that controls at least one operation, such a device may be implemented in hardware, firmware or software, or some combination of at least two of the same. It should be noted that the functionality associated with any particular controller may be centralized or distributed, whether

locally or remotely. Definitions for certain words and phrases are provided throughout this patent document, those of ordinary skill in the art should understand that in many, if not most instances, such definitions apply to prior, as well as future uses of such defined words and phrases.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention, and the advantages thereof, reference is now made to the following descriptions taken in conjunction with the accompanying drawings, wherein like numbers designate like objects, and in which:

FIGURE 1 is a block diagram of a processing system that contains a data processor in accordance with the principles of the present invention;

FIGURE 2 illustrates the exemplary data processor in greater detail according to one embodiment of the present invention;

FIGURE 3 illustrates a cluster in the exemplary data processor according to one embodiment of the present invention;

FIGURE 4 illustrates the operational stages of the exemplary data processor according to one embodiment of the present invention;

FIGURE 5 is a block diagram illustrating selected portions of an instruction fetch apparatus according to one embodiment of the present invention;

FIGURE 6 is a block diagram illustrating the contents of the instruction cache in the exemplary data processor according to one embodiment of the present invention; and

FIGURE 7A-7D are block diagrams illustrating the flow of instructions through the instruction issue buffers according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

FIGURES 1 through 7, discussed below, and the various embodiments used to describe the principles of the present invention in this patent document are by way of illustration only and should not be construed in any way to limit the scope of the invention. Those skilled in the art will understand that the principles of the present invention may be implemented in any suitably arranged data processor.

FIGURE 1 is a block diagram of processing system 10, which contains data processor 100 in accordance with the principles of the present invention. Data processor 100 comprises processor core 105 and N memory-mapped peripherals interconnected by system bus 120. The N memory-mapped peripherals include exemplary memory-mapped peripherals 111-114, which are arbitrarily labeled Memory-Mapped Peripheral 1, Memory-Mapped Peripheral 2, Memory-Mapped Peripheral 3, and Memory-Mapped Peripheral N. Processing system 10 also comprises main memory 130. In an advantageous embodiment of the present invention, main memory 130 may be subdivided into program memory 140 and data memory 150.

The cost and complexity of data processor 100 is minimized by excluding from processor core 105 complex functions that may be

implemented by one or more of memory-mapped peripherals 111-114. For example, memory-mapped peripheral 111 may be a video codec and memory-mapped peripheral 112 may be an audio codec. Similarly, memory-mapped peripheral 113 may be used to control cache flushing. The cost and complexity of data processor 100 is further minimized by implementing extremely simple exception behavior in processor core 105, as explained below in greater detail.

Processing system 10 is shown in a general level of detail because it is intended to represent any one of a wide variety of electronic devices, particularly consumer appliances. For example, processing system 10 may be a printer rendering system for use in a conventional laser printer. Processing system 10 also may represent selected portions of the video and audio compression-decompression circuitry of a video playback system, such as a video cassette recorder or a digital versatile disk (DVD) player. In another alternative embodiment, processing system 10 may comprise selected portions of a cable television set-top box or a stereo receiver. The memory-mapped peripherals and a simplified processor core reduce the cost of data processor 100 so that it may be used in such price sensitive consumer appliances.

In the illustrated embodiment, memory-mapped peripherals 111-114 are shown disposed within data processor 100 and program

memory 140 and data memory 150 are shown external to data processor 100. It will be appreciated by those skilled in the art that this particular configuration is shown by way of illustration only and should not be construed so as to limit the scope of the present invention in any way. In alternative embodiments of the present invention, one or more of memory-mapped peripherals 111-114 may be externally coupled to data processor 100. Similarly, in another embodiment of the present invention, one or both of program memory 140 and data memory 150 may be disposed on-chip in data processor 100.

FIGURE 2 is a more detailed block diagram of exemplary data processor 100 according to one embodiment of the present invention. Data processor 100 comprises instruction fetch cache and expansion unit (IFCEXU) 210, which contains instruction cache 215, and a plurality of clusters, including exemplary clusters 220-222. Exemplary clusters 220-222 are labeled Cluster 0, Cluster 1 and Cluster 2, respectively. Data processor 100 also comprises core memory controller 230 and interrupt and exception controller 240.

A fundamental object of the design of data processor 100 is to exclude from the core of data processor 100 most of the functions that can be implemented using memory-mapped peripherals external to the core of data processor 100. By way of example, in an exemplary

embodiment of the present invention, cache flushing may be efficiently accomplished using software in conjunction with a small memory-mapped device. Another object of the design of data processor 100 is to implement a statically scheduled instruction pipeline with an extremely simple exception behavior.

Clusters 220-222 are basic execution units that comprise one more arithmetic units, a register file, an interface to core memory controller 230, including a data cache, and an inter-cluster communication interface. In an exemplary embodiment of the present invention, the core of data processor 100 may comprise only a single cluster, such as exemplary cluster 220.

Because conventional processor cores can execute multiple simultaneously issued operations, the traditional word "instruction" is hereby defined with greater specificity. For the purposes of this disclosure, the following terminology is adopted. An "instruction" or "instruction bundle" is a group of simultaneously issued operations encoded as "instruction syllables". Each instruction syllable is encoded as a single machine word. Each of the operations constituting an instruction bundle may be encoded as one or more instruction syllables. Hereafter, the present disclosure may use the shortened forms "instruction" and "bundle" interchangeably and may use the

shortened form "syllable." In an exemplary embodiment of the present invention, each instruction bundle consists of 1 to 4 instruction syllables. Flow control operations, such as branch or call, are encoded in single instruction syllables.

FIGURE 3 is a more detailed block diagram of cluster 220 in data processor 100 according to one embodiment of the present invention. Cluster 220 comprises instruction buffer 305, register file 310, program counter (PC) and branch unit 315, instruction decoder 320, load store unit 325, data cache 330, integer units 341-344, and multipliers 351-352. Cluster 220 is implemented as an instruction pipeline.

Instructions are issued to an operand read stage associated with register file 310 and then propagated to the execution units (i.e., integer units 341-244, multipliers 351-352). Cluster 220 accepts one bundle comprising one to four syllables in each cycle. The bundle may consist of any combination of four integer operations, two multiplication operations, one memory operation (i.e., read or write) and one branch operation. Operations that require long immediates (constants) require two syllables.

In specifying a cluster, it is assumed that no instruction bits are used to associate operations with functional units. For example, arithmetic or load/store operations may be placed in any

of the four words encoding the operations for a single cycle. This may require imposing some addressing alignment restrictions on multiply operations and long immediates (constants).

This following describes the architectural (programmer visible) status of the core of data processor 100. One design objective of data processor 100 is to minimize the architectural status. All non-user visible status information resides in a memory map, in order to reduce the number of special instructions required to access such information.

Program Counter

In an exemplary embodiment of the present invention, the program counter (PC) in program counter and branch unit 315 is a 32-bit byte address pointing to the beginning of the current instruction bundle in memory. The two least significant bits (LSBs) of the program counter are always zero. In operations that assign a value to the program counter, the two LSBs of the assigned value are ignored.

Register File 310

In an exemplary embodiment, register file 310 contains 64 words of 32 bits each. Reading Register 0 (i.e., R0) always returns the value zero.

Link Register

Register 63 (i.e., R63) is used to address the link register by the call and return instructions. The link register (LR) is a slaved copy of the architecturally most recent update to R63. R63 can be used as a normal register, between call and return instructions. The link register is updated only by writes to R63 and the call instruction. At times the fact that the link register is a copy of R63 and not R63 itself may be visible to the programmer. This is because the link register and R63 get updated at different times in the pipeline. Typically, this occurs in the following cases:

1) ICALL and IGOTO instructions - Since these instructions are executed in the decode stage, these operations require that R63 be stable. Thus, R63 must not be modified in the instruction bundle preceding one of these operations. Otherwise unpredictable results may occur in the event of an interrupt; and

2) An interrupt or exception may update the link register incorrectly. Thus, all interrupt and exception handlers must explicitly write R63 prior to using the link register through the execution of an RFI, ICALL or IGOTO instruction. This requirement can be met with a simple MOV instruction from R63 to R63.

Branch Bit File

The branch architecture of data processor 100 uses a set of

eight (8) branch bit registers (i.e., B0 through B7) that may be read or written independently. In an exemplary embodiment of the present invention, data processor 100 requires at least one instruction to be executed between writing a branch bit and using the result in a conditional branch operation.

Control Registers

A small number of memory mapped control registers are part of the architectural state of data processor 100. These registers include support for interrupts and exceptions, and memory protection.

The core of data processor 100 is implemented as a pipeline that requires minimal instruction decoding in the early pipeline stages. One design objective of the pipeline of data processor 100 is that it support precise interrupts and exceptions. Data processor 100 meets this objective by updating architecturally visible state information only during a single write stage. To accomplish this, data processor 100 makes extensive use of register bypassing circuitry to minimize the performance impact of meeting this requirement.

FIGURE 4 is a block diagram illustrating the operational stages of pipeline 400 in exemplary data processor 100 according to one embodiment of the present invention. In the illustrated

embodiment, the operational stages of data processor 100 are address generation stage 401, fetch stage 402, decode stage 403, read stage 404, first execution (E1) stage 405, second execution (E2) stage 406 and write stage 407.

Address Generation Stage 401 and Fetch Stage 402

Address generation stage 401 comprises a fetch address generator 410 that generates the address of the next instruction to be fetched from instruction cache 215. Fetch address generator 410 receives inputs from exception generator 430 and program counter and branch unit 315. Fetch address generator 410 generates an instruction fetch address (FADDR) that is applied to instruction cache 215 in fetch stage 402 and to an instruction protection unit (not shown) that generates an exception if a protection violation is found. Any exception generated in fetch stage 402 is postponed to write stage 407. Instruction buffer 305 in fetch stage 402 receives instructions as 128-bit wide words from instruction cache 215 and the instructions are dispatched to the cluster.

Decode Stage 403

Decode stage 403 comprises instruction decode block 415 and program counter (PC) and branch unit 315. Instruction decode block 415 receives instructions from instruction buffer 305 and decodes the instructions into a group of control signals that are

applied to the execution units in E1 stage 405 and E2 stage 406. Program counter and branch unit 315 evaluates branches detected within the 128-bit wide words. A taken branch incurs a one cycle delay and the instruction being incorrectly fetched while the branch instruction is evaluated is discarded.

Read Stage 404

In read stage 404, operands are generated by register file access, bypass and immediate (constant) generation block 420. The sources for operands are the register files, the constants (immediates) assembled from the instruction bundle, and any results bypassed from operations in later stages in the instruction pipeline.

E1 Stage 405 and E2 Stage 406

The instruction execution phase of data processor 100 is implemented as two stages, E1 stage 405 and E2 stage 406 to allow two cycle cache access operations and two cycle multiplication operations. Exemplary multiplier 351 is illustrated straddling the boundary between E1 stage 405 and E2 stage 406 to indicate a two cycle multiplication operation. Similarly, load store unit 325 and data cache 330 are illustrated straddling the boundary between E1 stage 405 and E2 stage 406 to indicate a two cycle cache access operation. Integer operations are performed by integer units, such

as IU 341 in E1 stage 405. Exceptions are generated by exception generator 430 in E2 stage 406 and write stage 407.

Results from fast operations are made available after E1 stage 405 through register bypassing operations. An important architectural requirement of data processor 100 is that if the results of an operation may be ready after E1 stage 405, then the results are always ready after E1 stage 405. In this manner, the visible latency of operations in data processor 100 is fixed.

Write Stage 407

At the start of write stage 407, any pending exceptions are raised and, if no exceptions are raised, results are written by register write back and bypass block 440 into the appropriate register file and/or data cache location. In data processor 100, write stage 407 is the "commit point" and operations reaching write stage 407 in the instruction pipeline and not "excepted" are considered completed. Previous stages (i.e., address generation, fetch, decode, read, E1, E2) are temporally prior to the commit point. Therefore, operations in address generation stage 401, fetch stage 402, decode stage 403, read stage 404, E1 stage 405 and E2 stage 406 are flushed when an exception occurs and are acted upon in write stage 407.

As the above description indicates, data processor 100 is a

very large instruction word (VLIW) device that allow the parallel execution of multiple instructions in two or more instruction pipelines in clusters 220-222. In an exemplary embodiment, instruction cache 215 comprises cache lines that are 512 bits (i.e., 64 bytes) long. Each syllable (i.e., smallest instruction size) comprises 32 bits (i.e., 4 bytes), such that a cache line comprises 16 syllables. Each instruction syllable is encoded as a single 32-bit machine word.

Instructions are fetched from instruction cache 215 in groups of four syllables (i.e., 128 bits). A complete instruction may comprise one, two, three or four syllables. The fetched syllables are issued into one of four issues lanes leading into the instruction pipeline. The four issue lanes are referred to as Issue Lane 0, Issue Lane 1, Issue Lane 2, and Issue Lane 3. Because instructions are of variable length and because a branch instruction may fetch instructions starting at any point in instruction cache 215, there is no guarantee that all of the syllables in an instruction will be fetched in the same cache access. There also is no guarantee that a particular syllable in an instruction will be aligned to a particular issue lane in clusters 220-222.

In order to minimize the amount of delay incurred in fetching

instructions, the present invention implements an instruction issue unit comprising a sequence of instruction issue unit buffers (IIUBs) that temporarily store the syllables of an instruction until all syllables of the instruction are present. The complete instruction, consisting of one to four syllables, is then issued into the four issues lanes of the pipeline. If an instruction has less than four syllables, one or more no-operation (NOP) instructions are issued into the unused issue lanes. In the exemplary embodiment that follows, two instruction issue unit buffers are used to buffer up to four 32-bit syllables.

However, it should be understood that the selection of these values is by way of example only and should not be construed to limit the scope of the present invention. Those skilled in the art will recognize that other syllable size, buffer size and instruction sizes may be used. For example, in an alternate embodiment of the present invention, a syllable may comprise eight bits, sixteen bits, sixty-four bits, or the like, rather than thirty-two bits. Also, the instruction issue unit buffers may hold eight syllables, twelve syllables, sixteen syllables, or the like, instead of four syllables.

FIGURE 5 is a block diagram illustrating selected portions of instruction issue unit 500 according to one embodiment of the

present invention. Instruction issue unit 500 comprises instruction issue controller 550, registers 511, 521, 531 and 541, multiplexers (MUXs) 512, 522, 532 and 542, registers 513, 523, 533 and 543, and MUX 560. Registers 513, 523, 533, and 543 comprise a first instruction issue unit buffer, referred to hereafter as Instruction Issue Unit Buffer 0 (IIUB0). Registers 511, 521, 531, and 541 comprise a second instruction issue unit buffer, referred to hereafter as Instruction Issue Unit Buffer 1 (IIUB1).

The alignment of cache accesses to instruction cache 215 is determined by the branch target alignment. Each cache access after an access to a branch target fetches four syllables using the same alignment until the next taken branch occurs or a cache line boundary is crossed. Each line of the cache is organized as four independently addressable cache banks aligned with four issue lanes. The first cache bank holds Syllable 0 and is aligned with the first issue lane, referred to as Issue Lane 0. The second cache bank holds Syllable 1 and is aligned with the second issue lane, referred to as Issue Lane 1. The third cache bank holds Syllable 2 and is aligned with the third issue lane, referred to as Issue Lane 2. The fourth cache bank holds Syllable 3 and is aligned with the fourth issue lane, referred to as Issue Lane 3.

Since there is no guarantee that the first syllable of an

instruction is aligned to particular cache bank of issue lane, a branch address may access an instruction aligned starting in any issue lane and cache bank. Thus, a four syllable instruction may begin in the third cache bank (i.e., Syllable 2 position) and be aligned to Issue Lane 2. For example, if Instruction A comprises four syllables A0, A1, A2 and A3, the four syllables may be fetched into Issue Lane 2, Issue Lane 3, Issue Lane 0, and Issue Lane 1. A branch instruction is always indicated by the first syllable in an instruction bundle. Hence, the outputs of registers 513, 523, 533 and 543 are input to separate channels of multiplexer (MUX) 560 and are individually selected by the START OF BUNDLE control signal.

Instruction issue controller 550 controls the transfer of Syllable 3, Syllable 2, Syllable 1 and Syllable 0 from instruction cache 215 to Issue Lane 3, Issue Lane 2, Issue Lane 1, and Issue Lane 0, respectively. A Stop bit is use in the highest syllable of an instruction bundle to indicate the end of the syllable. Thus, in a three syllable instruction bundle comprising syllables A0, A1 and A2, the Stop bit is in syllable A2.

Ideally, each of the four syllables in an cache fetch are loaded from instruction cache 215 directly into the empty registers in instruction issue unit (IIU) buffer 0 (i.e., registers 513, 523,

533 and 543). In such a case, instruction issue controller 550 sets the MUX CONTROL signal to switch all four syllables to the inputs of registers 513, 523, 533 and 543. Instruction issue controller 550 also selectively enables each of registers 513, 523, 533 and 543 using individual Load Enable 2 (LE2) signals.

However, if previously fetched syllables are still in one or more of registers 513, 523, 533 and 543 when the next instruction bundle is fetched, instruction issue controller 550 sets the individual MUX CONTROL signals to selectively switch the corresponding ones of the four syllables in the next instruction to the inputs of registers 511, 521, 531 and 541 (i.e., Instruction Issue Unit (IIU) Buffer 1). Instruction issue controller 550 also selectively enables each of registers 511, 521, 531 and 541 using individual Load Enable 1 (LE1) signals. Thus, a syllable may be delayed temporarily in IIU Buffer 1 until the corresponding register in IIU Buffer 0 becomes empty.

The operation of instruction issue unit 500 may best be understood with reference to FIGURE 6 and FIGURES 7A-7D. FIGURE 6 is a block diagram illustrating the contents of instruction cache 215 in exemplary data processor 100 according to one embodiment of the present invention. FIGURE 7A-7D are block diagrams illustrating the flow of instruction bundles and syllables

through Instruction Issue Unit Buffer 0 (IIUB0) and Instruction Issue Unit Buffer 1 (IIUB1) according to one embodiment of the present invention.

Instruction cache 215 contains an exemplary sequence of seven instruction bundles, referred to as Instructions A, B, C, D, E, F and G within a single cache line. Instruction A comprises two syllables, A0 and A1. Instruction B comprises one syllable, B0. Instruction C comprises two syllable, C0 and C1. Instruction D comprises one syllable, D0. Instruction E comprises one syllable, E0. Instruction F comprises four syllables, F0, F1, F2 and F3. Finally, Instruction G comprises one syllable, G0.

Initially, IIUB0 and IIUB1 are empty and a branch instruction begins fetching syllables in groups of four beginning at syllable A0. Since IIUB0 is empty, instruction issue controller 550 sets MUX 512, MUX 522, MUX 532 and MUX 542 so that the first four syllables, A0, A1, B0 and C0, are fetched into IIUB0. Syllable A0 is in the Syllable 2 slot in FIGURE 5 and therefore is aligned with Issue Lane 2 (register 523). Correspondingly, Syllable A1 is aligned with Issue Lane 3 (register 513), Syllable B0 is aligned with Issue Lane 0 (register 543), and Syllable C01 is aligned with Issue Lane 1 (register 533).

FIGURE 7A shows the positions of A0, A1, B0 and C0 after they are loaded into IIUB0. After A0, A1, B0 and C0 are loaded into IIUB0, instruction issue controller 550 detects the Stop bit in A1, indicating that all of Instruction A has been fetched, and issues A0 and A1 into Issue Lanes 2 and 3. Syllables B0 and C0 remain in IIUB0. IIUB1 (i.e., registers 511, 521, 531 and 541) is still empty.

At this point, the next four syllables (C1, D0, E0 and F0) are fetched. Since IIUB0 is only partially empty, instruction issue controller 550 sets MUX 512 and MUX 522 so that syllables C1 and D0 are fetched into IIUB0 by the LE2 signal. Instruction issue controller 550 also sets MUX 532 and MUX 542 so that the syllables E0 and F0 can only be fetched into IIUB1 by the LE1 signal. FIGURE 7B shows the positions of C1, D0, E0 and F0 after they are loaded into IIUB0 and IIUB1. After C1, D0, E0 and F0 are loaded, instruction issue controller 550 detects the Stop bit in B0, indicating that all of Instruction B has been fetched, and issues B0 into Issue Lane 0. Syllables C0, C1 and D0 remain in IIUB0. IIUB1 contains E0 and F0.

At this point, the next four syllables (F1, F2, F3 and G0) are fetched. Since register 543 in IIUB0 is empty after syllable B0 is issued into Issue Lane 0, instruction issue controller 550 sets

MUX 542 so that syllable E0 is transferred from IIUB1 to IIUB0 by the LE2 signal. Instruction issue controller 550 also sets MUX 512, MUX 522 and MUX 542 so that the syllables F1, F2 and F3 are fetched into IIUB1 by the LE1 signal. The LE1 signal is not applied to register 531, which still holds syllable F0 from the previous fetch. Therefore, syllable G0 is not written to register 531 in IIUB1. FIGURE 7c shows the positions of E0, F1, F2, and F3 after they are loaded into IIUB0 and IIUB1. After E0, F1, F2 and F3 are loaded, instruction issue controller 550 detects the Stop bit in C1, indicating that all of Instruction C has been fetched, and issues C0 and C1 into Issue Lanes 1 and 2. Syllables D0 and E0 remain in IIUB0. IIUB1 contains F3, F0, F1 and F2.

At this point, the four syllables F1, F2, F3 and G0 are re-fetched in order to fetch G0, which was not loaded on the previous fetch. Since registers 533 and 523 in IIUB0 are empty after syllables C0 and C1 are issued, instruction issue controller 550 sets MUX 522 and MUX 532 so that syllables F0 and F1 are transferred from IIUB1 to IIUB0 by the LE2 signal. Instruction issue controller 550 also sets MUX 532 so that the syllable G0 is fetched into IIUB1 by the LE1 signal. The LE1 signal is not applied to registers 541 and 511, which still hold syllables F3 and F2 from the previous fetch. The LE1 signal is also not applied to

register 521, which is empty after syllable F1 is transferred to IIUB0. FIGURE 7C shows the positions of F0, F1 and G0 after F0, F1 and G0 are loaded into IIUB0 and IIUB1. After F0, F1 and G0 are loaded, instruction issue controller 550 detects the Stop bit in D0, indicating that all of Instruction D has been fetched, and issues D0 into Issue Lane 3. Syllables E0, F0 and F1 remain in IIUB0. IIUB1 contains F3, G0 and F2.

As FIGURE 6 and FIGURES 7A-7D demonstrate, instruction issue unit 500 continually fetches syllables as far "forward" as possible in IIUB0 and IIUB1. If IIUB0 and IIUB1 are empty, syllables are transferred directly into IIUB0, the "forward-most" instruction buffer. If a register in IIUB0 is not empty, the corresponding incoming syllable is instead loaded into IIUB1 and subsequently advances into IIUB0 when the corresponding register becomes empty. In alternate embodiments, one or more additional layers of buffering may be added by inserting additional banks of registers and multiplexers in front of IIUB0 and IIUB1.

By way of example, if a third layer of buffering is desired, a third instruction issue unit buffer, IIUB2, may be implemented by inserting a third register and a second multiplexer in each issue lane. For example, in Issue Lane 3, the output of the second multiplexer would be connected to the input of register 511, one

input channel of the second multiplexer would be connected to the output of the third register, and the other input channel of the second multiplexer would be connected directly to Syllable 3 output of instruction cache 215. The input of the third register also would be connected directly to Syllable 3 output of instruction cache 215. The second multiplexer and the third register would be controlled by instruction issue controller 550 using a second multiplexer control signal (MUX CONT. 2) and a third load enable signal (LE3). Those skilled in art will recognize that the present invention may be similarly extended to implement additional layers of instruction issue buffers (i.e., IIUB3, IIUB4, IIUB5 and so forth).

Although the present invention has been described in detail, those skilled in the art should understand that they can make various changes, substitutions and alterations herein without departing from the spirit and scope of the invention in its broadest form.